Projeto de Sistemas Operativos 2018-19
`CircuitRouter-Bench`
Visão geral do projeto

LEIC-A / LEIC-T / LETI
IST

O projeto de Sistemas Operativos 2018/19 está organizado em dois exercícios não avaliados para familiarização com o problema e com as ferramentas, seguidos de uma série de 3 exercícios com avaliação. Esta secção introdutória apresenta uma visão global do projeto e dos tópicos abordados em cada um dos exercícios.

O projeto consiste em desenvolver um serviço chamado `CircuitRouter-Bench`, que recebe pedidos com diferentes especificações para circuitos integrados bidimensionais ou tridimensionais contendo múltiplas interconexões. Para cada pedido, o `CircuitRouter-Bench` calcula a melhor opção para posicionar cada interconexão no circuito de forma a evitar sobreposições entre as interconexões.

Para circuitos de dimensão e complexidade (número de interconexões) elevados, este problema pode ser muito demorado quando resolvido de forma sequencial. Assim sendo, o `CircuitRouter-Bench` fará uso de técnicas de programação concorrente que serão aprendidas ao longo do semestre de Sistemas Operativos para acelerar substancialmente a execução aproveitando o paralelismo *hardware* disponível nas máquinas *multi-core* atuais.

1 Problema do roteamento em circuitos

Considere-se um circuito integrado ou impresso, composto por vários componentes eletrónicos, posicionados no circuito. Por exemplo, os componentes podem ser os transístores num circuito integrado, ou os componentes dispostos na placa mãe (*motherboard*) de um PC. Estes componentes são interligados numa grelha bi ou tri-dimensional.

O problema do roteamento em circuitos consiste em desenhar, de forma automática, os caminhos que interligam as componentes do circuito.

Na sua forma mais simples, o problema pode ser reduzido ao problema de juntar pontos numa grelha bi-dimensional que representa o circuito. Dados dois componentes que se pretendem ligar num circuito, o algoritmo de Lee [1] encontra o caminho mais curto entre os componentes usando a técnica ilustrada na Figura 1.

Começando no ponto de origem, os pontos da grelha são numerados por expansão da frente de onda até ao ponto de destino ser alcançado (Figura 1(a)-(d)). Em cada etapa da expansão, cada ponto p da frente de onda seleciona os pontos vizinhos que ainda não estejam numerados e numera-os com o incremento do valor de p. Assim que o ponto de destino é alcançado (ou seja, é numerado por um vizinho), o caminho mais curto é selecionado regressando ao ponto origem através de uma sequência de pontos com valores decrescentes (*backtracking*), tal como ilustrado na Figura 1(e).

Na prática, muitos dos pontos da grelha de interligações de um circuito poderão estar ocupados, por exemplo, por componentes ou por outros caminhos previamente definidos (cujos pontos não podem ser usados por outros caminhos). A fase de expansão do algoritmo tem este aspeto em conta, expandindo-se

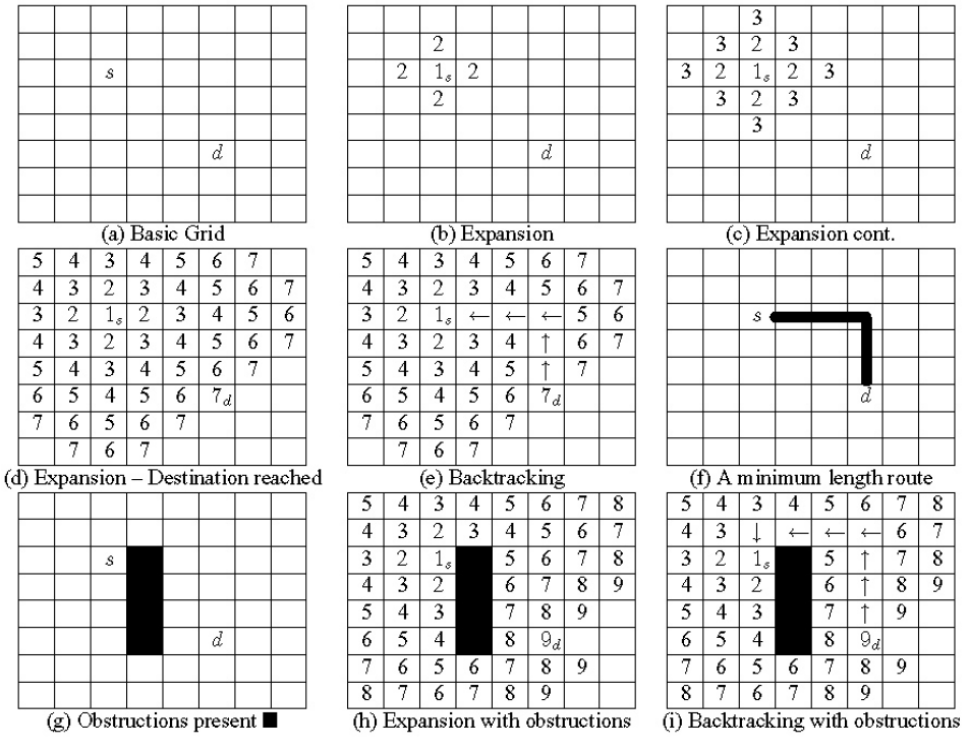

Figura 1: Diferentes exemplos a ilustrar as fases de expansão e de escolha regressiva (*backtracking*) do caminho no algoritmo de Lee (fonte: [2])

apenas em pontos livres e contornando os pontos já ocupados. Os exemplos nas Figuras 1(g)-(i) ilustram um exemplo disto, mostrando como o algoritmo de Lee consegue, apesar da existência de pontos já ocupados (logo inutilizáveis), contornar essas obstruções e encontrar o caminho mais curto possível. O pseudo-código do algoritmo de Lee está apresentado no Algoritmo 1.

Algoritmo 1: Pseudo-código do algoritmo de Lee

```
entrada: grelha e rotas
início
    para cada r em rotas faça
        origem, destino ← r
        marcar ponto origem com valor 0
        p ← 0

        /* fase de expansão da frente de onda desde a origem até ao destino */
        repita
            na grelha, marcar os vizinhos livres dos pontos de valor p com p + 1
            p ← p + 1
        até atingir destino ou não haver pontos livres

        /* retroceder do destino até à origem seguindo pontos de valor decrescente */
        selecionar ponto de destino
        repita
            adicionar ponto ao caminho
            selecionar o ponto vizinho de menor valor que o atual
        até atingir origem
        adicionar origem ao caminho

        na grelha, marcar pontos do caminho definido como ocupados
        na grelha, limpar valores dos pontos restantes
```

Um refinamento importante que pode ser feito ao algoritmo base consiste em aproveitar o facto de muitos circuitos reais permitirem mais do que uma camada de interligações. Estas camadas permitem a um caminho sobrepor-se a um obstáculo (por exemplo, um outro caminho) desviando-se uma camada acima do obstáculo. Este refinamento é facilmente incorporado no algoritmo acima se se considerar a grelha como sendo tri-dimensional, em que a 3ª dimensão corresponde ao número de camadas suportado.

Um outro refinamento relevante suportado pelo algoritmo consiste em atribuir custos diferentes às várias direções que podem ser escolhidas para traçar uma dada interligação. Por exemplo, por omissão, uma interligação na direção do eixo z (i.e., que atravessa duas camadas dum circuito tridimensional) tem um custo duas vezes maior do que uma interligação ao longo dos eixos x ou y (pois estas interligações assentam na mesma camada). Adicionalmente, para minimizar o número de "curvas" (i.e., variações na direção escolhida para traçar uma determinada interligação) o algoritmo de roteamento considera também um *bending cost*, ou seja, cobra um custo adicional cada vez que é considerada uma opção de roteamento que obriga a mudar de direção.

A aplicação `CircuitRouter-Solver` resolve o problema do roteamento em circuitos aplicando o algoritmo de Lee, descrito acima. Mais concretamente, o `CircuitRouter-Solver` tem como ponto de partida uma grelha tri-dimensional de dimensões (comprimento, largura, altura) pré-conhecidas, que pode ter alguns pontos inicialmente ocupados. Além do estado inicial, o `CircuitRouter-Solver` recebe uma lista de pontos (origem, destino) que devem ser interligados pelo caminho mais curto. Na maioria das vezes, é possível encontrar o caminho entre um par de pontos, mesmo quando há obstáculos a contornar. No en-

tanto, excepcionalmente, pode haver pares de pontos para os quais não exista nenhum caminho possível, pois, devido aos obstáculos existentes na grelha, o ponto origem está isolado do ponto destino; nesses casos, o `CircuitRouter-Solver` descarta o par (origem, destino) problemático e passa ao próximo par na lista.

2 Organização do projeto

O ponto de partida do projeto será uma implementação sequencial do `CircuitRouter-Solver`, com o nome `CircuitRouter-SeqSolver`, que implementa o algoritmo descrito na secção anterior e cujo código será fornecido no site da disciplina.

Esse projeto inicial será gradualmente estendido ao longo de 4 exercícios, que são aqui apresentamos de forma resumida:

- Exercício 0: Os guiões das duas primeiras aulas de laboratório (semanas 17-21/setembro e 24-28/setembro) compõem o exercício 0, que não é avaliado. O objetivo deste exercício é familiarizar os alunos com a solução inicial do `CircuitRouter-SeqSolver` e com as ferramentas de compilação, depuração e *scripting* que serão usadas no projeto `CircuitRouter-Bench`.
- Exercício 1: Desenvolverá a versão simples da consola `CircuitRouter-Shell`, denominada `CircuitRouter-SimpleShell`, que permite lançar e gerir instâncias `CircuitRouter-SeqSolver`. Cada instância do `CircuitRouter-SeqSolver` calcula a solução para um problema distinto submetido pelo utilizador, sendo que as diferentes instâncias lançadas executar-se-ão em paralelo (entre si) para melhor desempenho. O `CircuitRouter-SeqSolver` será também estendido para usar ficheiros para receber os dados de entrada e imprimir os resultados.
- Exercício 2: Construirá uma versão multi-tarefa concorrente do `CircuitRouter-Solver`, denominada `CircuitRouter-ParSolver`, recorrendo às técnicas de sincronização ensinadas na disciplina, que aproveitará o paralelismo do sistema para acelerar a execução de cada instância.
- Exercício 3: Estenderá o projeto com uma coordenação mais sofisticada entre a `CircuitRouter-Shell` e o `CircuitRouter-Solver`, recorrendo a mecanismos de coordenação/comunicação entre processos, correspondendo às versões `CircuitRouter-AdvShell` e `CircuitRouter-ParSolver`.

3 Entrega e avaliação

Esta secção aplica-se apenas às entregas dos exercícios com avaliação (exercícios 1 a 3).

Para cada exercício com avaliação, os alunos devem submeter um ficheiro no formato zip com o código fonte e o ficheiro *Makefile*. O arquivo submetido não deve incluir outros ficheiros tais como binários. O exercício deve **obrigatoriamente** compilar e executar nos computadores dos laboratórios.

As datas limite para a entrega de cada exercício com avaliação são as seguintes:

- Exercício 1: 12/outubro, 23h59 – `CircuitRouter-SimpleShell` + `CircuitRouter-SeqSolver`
- Exercício 2: 12/novembro, 23h59 – `CircuitRouter-SimpleShell` + `CircuitRouter-ParSolver`
- Exercício 3: 30/novembro, 23h59 – `CircuitRouter-AdvShell` + `CircuitRouter-ParSolver`

A submissão é feita através do Fénix.

Referências

[1] Rubin, F.. “The Lee Path Connection Algorithm.” IEEE Transactions on Computers C-23 (1974): 907-914.

[2] Watson, Ian et al. “A Study of a Transactional Parallel Routing Algorithm.” 16th International Conference on Parallel Architecture and Compilation Techniques (PACT 2007) (2007): 388-400.

Nota: Não é necessário consultar as referências acima para resolver o projeto. As referências são facultadas apenas para leitura futura pelos mais curiosos.